02

Más sobre polimorfismo

Transcripción

[00:00] Bienvenidos a una parte más de su curso de Java. Haciendo un overview sobre lo que hemos hablado en la clase anterior, ya sabemos los conceptos básicos de herencia, polimorfismo, sabemos ya los fundamentos detrás de estos conceptos, y en esta clase lo que vamos a hacer es profundizar un poco más en ellos.

[00:21] Para esto yo voy a hacer uso de una clase y de nuestro proyecto del curso anterior, que está aquí. Pero bueno, voy a comenzar explicando un poco la motivación. Ya sabemos que estamos haciendo un proyecto de un banco, y en el banco aparte de funcionarios, gerentes y eso, tenemos lo básico que es la cuenta.

[00:44] Ya en el curso anterior definimos la estructura de nuestro banco en base a la cuenta y lo vamos a ver justo ahora, lo tengo justo aquí, abro el proyecto y vamos a ver qué tiene aquí. Listo, vemos que aquí está nuestra clase cuenta, abrimos, y la cuenta ya recordando un poco pues teníamos saldo, agencia, número y el titular que era un objeto del tipo cliente.

[01:13] A su vez nuestro cliente tenía nombre, documento y teléfono. Para no tener que escribir todo esto de nuevo, vamos a usar la misma clase, entonces lo que voy a hacer es darle cuenta, copiar, voy a crear cuenta, un "Ctrl + C" y aquí un "Ctrl + V". Y listo, copié. Y vemos que me da un error de compilación. ¿Por qué?

[01:35] Porque falta la clase cliente. Recordemos que la cuenta tiene dentro un objeto cliente que no está aquí en este proyecto. Bueno, se soluciona fácil, "Ctrl + C" a cliente, "Ctrl + V" aquí y listo, asunto arreglado, tenemos nuestro código compilando y está todo en orden. Ahora, como este proyecto no lo vamos a utilizar más, lo que voy a hacer es cerrarlo. Entonces cierro aquí, clic derecho y close project. Perfecto, listo.

[02:08] Ahora que ya tenemos nuestra clase cuenta lista en este nuevo proyecto, vamos a definir una nueva regla del negocio para nuestro Byte Bank heredado, para nuestro banco. Sabemos que si bien tenemos en un banco la cuenta, hay tipos de cuenta. Al igual que hay tipos de funcionario, nosotros tenemos tipos de cuenta. Puede ser cuenta de ahorro, cuenta corriente, cuenta sueldo y etcétera.

[02:42] ¿Entonces qué es lo que debemos hacer ahora o cuál es nuestro propósito? Comenzar a definir esos tipos de cuenta de la misma forma que lo hemos hecho dentro de funcionario, contador y etcétera. Para eso vamos a hacer uso pues de herencia y polimorfismo. Vamos a darle aquí a New class y vamos a comenzar con nuestra cuenta corriente.

[03:08] Perfecto. Esa cuenta corriente va a extender de nuestra clase cuenta, perfecto. Entonces aquí tenemos el botón Superclass. ¿Superclass qué nos dice? De qué clase él va a extender o de qué clase él va a ser hijo. Entonces le damos un browse. Por defecto, siempre todas las clases extienden de object pero no lo queremos en ese caso entonces vamos a escribir cuenta.

[03:34] Y vemos que él detecta automáticamente a clase cuenta. Seleccionamos, le damos finish y él ya está creando nuestro código public class CuentaCorriente extends Cuenta. Es lo que necesitábamos. Pero él no está compilando aquí y no dice que el problema es que el constructor, por defecto cuenta está indefinido. ¿Qué significa eso? Que el constructor sin parámetros de la clase cuenta no existe y por eso él no puede crear esta clase tranquilamente.

[04:11] Vamos a ir a la clase cuenta, solamente para recordar un poco y habíamos quedado que yo no podía crear una cuenta, si no le daba un número de agencia y un número. ¿Cierto? Entonces, si yo quiero solucionar este problema de aquí, la primer solución que yo podría implementar es un constructor por defecto. Por ejemplo es público. Listo.

[04:39] Ahora que he creado un constructor por defecto, vemos que compila sin errores. ¿Por qué? Porque ya está añadido el constructor por defecto. ¿Soluciona el problema? Sí, pero rompe con nuestra regla del negocio de que la cuenta necesita sí o sí una agencia y un número para ser creado, entonces no es posible que usemos este constructor por defecto, no podemos romper las reglas del negocio.

[05:09] Entonces borramos, guardamos, vemos que nuevamente él ya no está compilando y aquí viene lo interesante. Vamos a crearle un constructor para nuestra clase CuentaCorriente. Y ya sabemos que si nosotros llamamos a super, llamamos al constructor de la clase padre, pero ya que no tiene constructor por defecto sin parámetros, recordemos que super llama al constructor por defecto.

[05:45] O perdón, mejor dicho, super llama a cualquier atributo de la clase padre, y si lo llamamos así nada más, llamamos al constructor por defecto. Pero si no lo tiene, puede ser la condición. Vamos a verlo de otra forma. Si mi clase cuenta, que es mi clase padre, me exige siempre especificar una agencia y un número para que pueda existir, ¿no creen que la clase hija debería seguir la misma regla, ya que es una restricción de la clase padre?

[06:18] Entonces si yo necesito que mi clase hija cumpla con el mismo requerimiento, voy a mandarle estos mismos parámetros aquí a la cuenta corriente, entonces cada vez que yo cree una cuenta corriente, le voy a tener que enviar también una agencia y un número, que es lo que corresponde, y aquí en el constructor le voy a mandar como parámetro la agencia y el número.

[06:47] Se fue aquí el mouse. Y el número. Y ahora, el código compila correctamente. ¿Por qué? ¿Cómo es que funciona esto? Vamos a dar otro review aquí. Recordemos que super lo que hace es acceder a los métodos de la clase padre, y en la clase padre tenemos este constructor. Entonces super lo que está haciendo es llamar a este constructor usa estos dos parámetros, que en este caso es este de aquí.

[07:21] Si nosotros tuviéramos un constructor por defecto, vamos a volver a crearlo aquí. Voy a borrar esta mayúscula que está mal. Si tuviéramos este constructor vacío, entonces super nos permitiría llamar al constructor por defecto y el código seguiría compilando, pero como no es el caso porque necesitamos satisfacer las reglas del negocio, nuevamente borramos esto y de esta forma accedemos ya al constructor de la clase padre y mantenemos la integridad de la cuenta.

[08:00] Y de la misma forma que lo hemos hecho para la cuenta corriente, vamos a crear nuestra cuenta de ahorros. Entonces vamos a hacer el mismo proceso, new class y vamos a llamar aquí CuentaAhorros. Perfecto. Y nuevamente browse, escribimos aquí cuenta para llamar a esa clase, y vemos que ahora está CuentaCorriente y Cuenta. Seleccionamos cuenta y le damos finish.

[08:29] Y de la misma forma. Pero aquí Eclypse ya aquí me está dando una pequeña ayuda más. ¿Cuál es? Si vemos bien que Eclypse reporta aquí el error que se está presentando, aquí abajo él me está dando la solución que es agregar constructor cuenta de ahorros y automáticamente él me genera el código que yo escribí aquí manualmente, él me lo autogenera aquí automáticamente pues seteando los dos parámetros y enviando al constructor de la clase padre.

[09:11] Y aquí, esta anotación TODO me indica pues que este código fue autogenerado por el IDE. Lo voy a borrar y ya tenemos nuestras dos clases, cuenta de ahorro y cuenta corriente, listas para trabajar e implementar nuestros siguientes ejemplos.